PPIG 2015: The impact of Syntax Highlighting in Sonic Pi

Giovanna Maria Dimitri[1]

Computer Laboratory, University of Cambridge
gmd43@cam.ac.uk

Abstract. We present an empirical study investigating the role of syntax highlighting in Sonic Pi, considering both program writing and program debugging tasks. Data were collected from 10 participants, who were asked to execute writing and debugging tasks, while their screens were recorded. We observed how syntax highlighting significantly improves task completion time in the case of writing as well as of debugging tasks. In particular we investigated debugging of a common mistake in the Sonic Pi environment, which we called colon-space feature because of the underlying Ruby syntax. Moreover we observed a correlation between syntax mistakes with programming experience and musical experience. If the task is highlighted the task completion time decreases with the increasing of programming experience, while this decreasing trend is not so significant in the case of an increasing musical experience.

Keywords: POP-II B. Coding; POP-II B. Debugging; POP-III D. Sonic Pi; POP-III C. Syntax Highlighting; POP-III D. Editors;

1 Introduction

Syntax highlighting is a feature of some text editors, that colour lexical elements in the source code, according to some semantic categorisations. It is used in source code text editors due to its capability of making the coding action easier. Moreover it should be easy to find certain types of code mistakes with syntax colouring. It is important to point out how syntax highlighting represents just a secondary notation, and has no effect on the behaviour of the code itself (Sarkar, 2015). The purpose of my study is to evaluate the impact of Syntax Highlighting in coding within the Sonic Pi environment, a new open source software tool and platform for the Rasperry Pi computer (Aaron, Blackwell, & Burnard, in review) and now also freely available for Windows and Mac distributions, designed for encouraging the learning of computing and music within schools (*Sonic Pi Website*, 2015). An example of how the Sonic Pi source code looks differently, with and without syntax highlighting, can be found in Figure 2 and Figure 1.

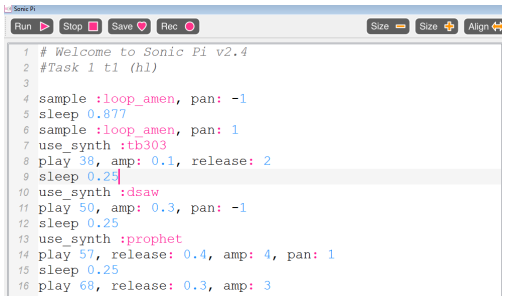

Fig. 1: Sonic Pi highlighted text editor

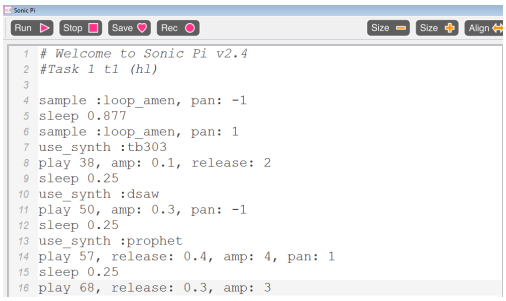

Fig. 2: Sonic Pi non-highlighted text editor

The main goal of Sonic Pi project was to create a new DSL (Domain Specific Language), having a double goal in terms of providing an engaging musical experience for students, and teaching them the basic elements of programming. The project was developed for a target class of 12 years old students and for 5 one-hour lessons, starting from teaching them what is a computer, up to coding (Aaron & Blackwell, 2013).
Sonic Pi was implemented as an embedded Ruby DSL language (Aaron & Blackwell, 2013). To make an example of Sonic Pi code, to trigger a new sound and then create a pause of a certain duration, before the next sound, the commands are:

```
play 60
sleep 0.5
play 61
```

The command in the first line will play the MIDI note 60 on the synthesiser used at that particular time, then a pause (whose duration is defined by the numerical number associated to sleep, in this case 0.5, that means half second) and then again another note.
To allow children to be able to code with Sonic Pi music language, a specific IDE was designed. The IDE is characterized by a scheme consisting of only 5 elements (Control buttons, Workspace tabs, Editor pane, Information pane, Error pane) and represents a simplified interface, suitable for children and non-expert teachers (Aaron & Blackwell, 2013).
No previous studies focused on the role of syntax highlighting in terms of speed and readability of the code in the Sonic Pi environment. We measured the time needed to complete an assigned musical-writing task in Sonic Pi and the time needed to debug a code in Sonic Pi. Speed in live coding has in fact a double meaning, as explained in (Zmölnig & Eckel, 2007). On the one hand it is interesting to know how fast a programmer/composer can write codes, since it is a way to measure its productivity in terms of line of codes produced in a certain amount of time. On the other hand, often writing code fast is a matter of virtuosity, as it is playing fast for a music performer. Live coding, in fact, quite naturally refers to some notion of performance, and speed is a performance factor which should surely be taken into account in its analysis.
We describe the experimental setting in Section 2. The research objectives are written as hypothesis in Section 3 and our findings described in Section 4.

1.1 Background

In the psychology literature there are many works focusing on the role of colouring tokens in prose texts. Van Nes (Van Nes, 1986) described the importance of colours, layouts and typography on display screens in order to create texts with the highest possible legibility.
Hakala (Hakala, Nykyri, & Sajaniemi, 2006) presented an experiment using three different colouring schemes for intermediate programmers, asked to look for local patterns in Java programs. The differences among colouring schemes proposed were not statistically significant, and the results showed how the classical used control scheme (black text on white background colour) exhibited the same performance in searching as the other control schemes.
Baecker (Baecker, 1988) developed a number of new techniques and tools to improve program source texts and program documentation. In particular he proposed a new effective presentation of source text in C programming language, using high quality digital typography and a processor implementing the design.
The main empirical study on syntax highlighting, which can be considered as a starting point for our work, is (Sarkar, 2015).
Sarkar (Sarkar, 2015) analysed the impact of syntax highlighting on program comprehension and the relation with programming experience of the subject. His studies were validated using eye-tracking data coming from 10 participants. The task submitted to each participant was to compute a Python function given a set of arguments. The python function was given several times with both highlighted and non-highlighted code. The results of the research study showed how the code with syntax highlighting reduced the task completion time, even if this effect decreased when participants were experienced programmers. Moreover it was shown how syntax highlighting improved the ability of the programmer to mentally retain the state of the execution, suggesting how a highlighted code brings to a lower mental comprehension effort (Sarkar, 2015).

2 Experimental Methodology

As participants we recruited 10 students of the University of Cambridge. The target students we looked at had to have some interest in music and programming. The variance of music and programming experience among the participants was quite high, from students who used programming everyday to students who just used programming for some courses. The same for the musical experience, some students had high musical experience, while others were listening to music as a hobby. The participants were recruited through the Department of Computer Science and college network.
The experiment procedure was composed by the following phases described in the following sections.

2.1 Tutorial Phase

The participants were given the Sonic Pi tutorial and a worksheet with some Sonic Pi examples to read. The worksheet was chosen from the Sonic Pi website, since it is a well built summary of the main topics of the tutorial and presented examples that could be useful to the participants to solve the tasks proposed. The maximum amount of time they could spend in reading the tutorial was 45 minutes. They were allowed to ask questions in case they needed further explanations. We decided that the tutorial was the best way to teach participants about Sonic Pi, to avoid the instructor bias while they were learning Sonic Pi basic commands. The tutorial was given to them in paper format, and they were not allowed to experiment the commands on the Sonic Pi software. This decision was taken because otherwise they would have tried either on the highlighted or non-highlighted version, and outcomes could have been biased by this. The printed tutorial had the same highlighting format as the one in Sonic Pi. Yet, most participants noticed how the highlighting of the syntax in the tutorial does not correspond to how the text is highlighted in the Sonic Pi text editor. This may be something that could be useful to change in the Sonic Pi tutorial, to make the learning process easier by having corresponding colours in the tutorial and the text editor. The tutorial and the worksheet were available to them also during the execution of the tasks, so they didn't need to memorize instructions.

2.2 Writing Task Phase

After the tutorial phase participants were asked to complete the writing task. To allow a within-subject comparison, each participant was asked to write two pieces of code having comparable difficulties, where one of the codes was highlighted and the other plain. The two programs were different and submitted to them sequentially. The same program could not be used for both cases since once a task is performed, it may be considerably easier to repeat the experiment with syntax highlighting (or non-highlighted text editor), just because the code is already known. Moreover the participants were divided in two groups (Group A and Group B). Group A was asked to write the first piece of code in the highlighted environment and the second in the non-highlighted one, while Group B was asked to complete the tasks in the reverse order. This scheme was adopted to prevent results being affected by order effects.
We designed the instructions for the task pairs carefully, describing step by step the piece of code requested, submitting two tasks of comparable difficulties, yet different.
These differences included the use of different coding structures (blocks in one and threads in the other, for example) having as a consequence little transferable knowledge between one and the other, in so doing minimising the order effects. The two pieces of code asked to write were two famous children songs (Brother John and Happy Birthday). The participants were given the instructions, and at the end they were asked to run them and try to recognize which song they had composed. This element of fun was introduced to let the participant enjoy even further the tasks requested and the musical aspect of coding with Sonic Pi. See Appendix for the task instructions.

2.3 Debugging Task Phase

Sonic Pi music language was built on the Ruby syntax, and many Ruby syntax conventions have been inherited. In particular the aspect analysed in our experiments is the role that the highlighted syntax code plays in understanding where to introduce a space after the colon in the code. This kind of colon-space feature is typical of Ruby. This peculiarity of Ruby regarding colon and spaces seems to be an issue also among Ruby users, as some coding forums report (*Coding Forum 1*, n.d.), (*Coding Forum 2*, n.d.), (*Coding Forum 3*, n.d.).
Consider the example below:

```
play 48,amp: 0.5
sample :ambi_piano
```

First of all, as we can see from the sampled two lines, all the numerical values in the Sonic Pi highlighting are blue, while all the strings are pink, and also the colon sign is pink.
In the first line the colon has to be followed by a space, because it means that we are assigning a certain value (0.5) to the amp (amplitude of a key) parameter. On the other hand, in the second line we have that the space is located before the colon, because we are instantiating a particular type of sample, which is ambi_piano in this case, and not assigning a specific value to a certain parameter. There are various considerations to be made on the way the code is highlighted, and how it works, that I have noticed while using Sonic Pi:

1. If in the first line we don't insert a space, between the variable name and the value we want to assign, the code still works but the syntax highlighting varies. In particular it becomes:

```
amp:0.5
```

2. If in the first line we insert a space, between the name of the variable and the value we want to assign, then the code doesn't work and the syntax highlighting of the code becomes:

amp :0.5

3. If in the second line we don't insert the space before the colon, the code still works correctly even if the difference is that no auto-completion possibility is given to the user.

sample:ambi_choir

4. If in the second line we insert the space after the colon, then the code doesn't work, and the syntax highlighting becomes the following:

sample: ambi_choir

To evaluate the role of syntax highlighting, in the colon-space scheme, participants were asked to debug a code. All the mistakes (6 in total) were related to the colon-space scheme sparse within the code, and presenting all the possible configurations of the mistakes so that they were not easily recognizable by the participants. Not all the colon-space schemes included in the code were wrong, because otherwise the task would become too easy and it wouldn't reflect what happens in reality.
As previously described in the writing phase participants were asked to debug pairs of codes: one highlighted and the other non-highlighted. Moreover, as in the previous phase participants were divided in two groups and all the considerations regarding preventing order effects and the comparable difficulties of the two pieces of codes designed can be repeated. See the Appendix for the task instructions.

2.4 Questionnaire

At the end of the experiments all the participants were asked to complete a questionnaire to understand their programming skills as well as their music knowledge and experience. In particular following (Sarkar, 2015) the questionnaire included specific questions regarding the musical and programming experience.

1. Do you have any programming experience?
2. If yes which language are you mostly familiar with?
3. How long have you been using that particular language?
4. How often have you used this ?
5. Which is the largest or most complex program you have written with this?
6. Self-report a score form 1-10 assessing your experience, with 10 being highly experienced

1. Do you have any kind of musical experience?
2. Do you play any instruments?
3. If yes which one?
4. Self-report a score from 1-10 assessing your musical experience with 10 being highly experienced in music.

These questions have been used in order to understand the impact of programming and musical experience in the performances of the various participants. In particular, they were helpful to rank participants by experience, since evaluating programming and music experience is extremely difficult to achieve, and the self-questionnaire is one of the better methods to build such ranking, even if also this can be subject to self-bias (over/under-estimation of individual programming-music capacities). That is why question 6 for the programming and question 4 for the musical experience was used to make a pairwise comparison between participants, while the other questions provided enough information to adjust unrealistic answers.

2.5 Procedure

We conducted the study in an isolated environment, with no distractions, so that all the participants were able to focus on the tasks proposed. To screen, and voice recording the experiments, we used Camtasia Studio (*Camtasia Studio*, n.d.) that allows to record the screen and present a studio software permitting to watch frame by frame the video recorded. This allowed exact evaluation of the timings needed to complete the tasks.
The experimental procedure was explained to the participants through an introductory speech. They were informed that all the data collected were going to be anonymised and they were asked to agree on the recording of the screen, and on the voice recording present in the room.
The tutorial phase timing never reached 45 minutes, and all the participants completed their reading within the time allowed (on average 20 minutes). The instructions of the tasks were submitted sequentially and 5 minutes were given to each participant before the beginning of each task, in order to read the instructions.
The timing data collected were all considered from the starting of the typing till when the participant typed the last command on the text editor. Participants were allowed to vocalize their mental processes, or to use paper and pen in case they needed to write some notes while doing the task.

2.6 Variables and Hypotheses

The variables studied are presented in Table 1. Then for evaluating the programming experience with respect to the task completion times two more derived variables were introduced, that are not included in this table. Based on our research questions, we formulated the 4 null hypothesis in Table 2. The first 2 relate to task completion times, while the last two relate to the relation with programming and musical experience. The hypotheses just assume that the distribution is not the same between the highlighted and non-highlighted cases. The significance level adopted throughout the analysis is α=0.05 and any result reported as significant have a p value below this. The test used for establishing normality is the Shapiro-Wilk test.

Table 1: Variables

Highlighting	Independent variable, binary and categorical, that is associated with the presence or not of highlighted text editor (HL, NHL)
Task Completion Time	A dependent, continuous variable, indicating the amount of time needed to a participant to complete a task (writing or debugging). The time is expressed in the format min.seconds
Programming Experience	An independent ordinal variable. This variable assigns a value between 1 and 10 to the programming experience of the participant, with 10 being most experienced
Musical Experience	An independent ordinal variable. This variable assigns a value between 1 and 10 to the programming experience of the participant, with 10 being most experienced

The null hypotheses are presented in Table 2.

3 Results

3.1 Results of Highlighting on the writing task completion time

The task completion times for highlighted as well as non-highlighted writing task with Sonic Pi were normally distributed. Therefore we performed a paired t test. The resulted p value shows how the highlighted version of the tasks were significantly faster than the non-highlighted ones. Consequently we reject $H1_0$.

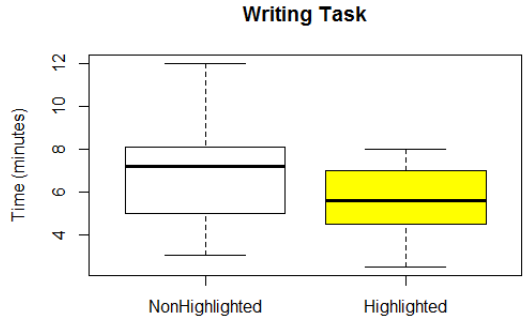

Fig. 3: Boxplots comparing Task 1 highlighted and non-highlighted instances

Table 2: Null Hypotheses

$H1_0$	There is no significance difference, in the writing task with Sonic Pi, of the completion time, between code with highlighting and code without
$H2_0$	There is no significance difference in the debugging task in Sonic Pi completion time, between code with highlighting and code without
$H3_0$	The effect of highlighting on task completion time of the writing task is not related to programming experience
$H4_0$	The effect of highlighting on task completion time of the writing task is not related to musical experience
$H5_0$	The effect of highlighting on task completion time of the debugging task is not related to programming experience
$H6_0$	The effect of highlighting on task completion time of the debugging task is not related to musical experience

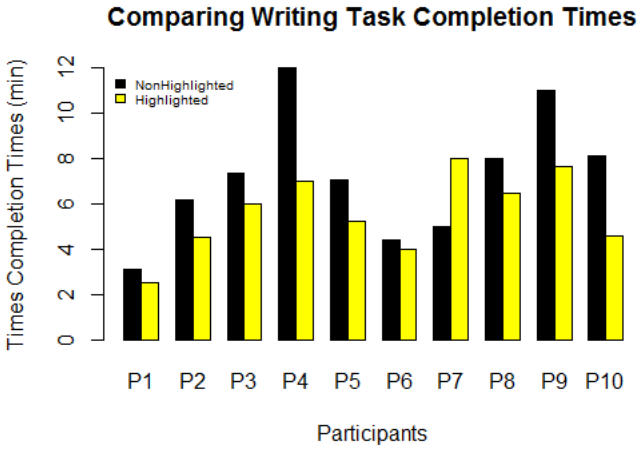

Fig. 4: Bar Graph comparing task completion times for a highlighted writing task with its non-highlighted counterpart

3.2 Results of Highlighting on the debugging task completion time

The task completion times for highlighted as well as non-highlighted debugging task with Sonic Pi were not normally distributed. Therefore the comparison was made using the Wilcoxon signed-rank test which shows that the highlighted version of the tasks were significantly faster than the non-highlighted version. Consequently we reject $H2_0$.

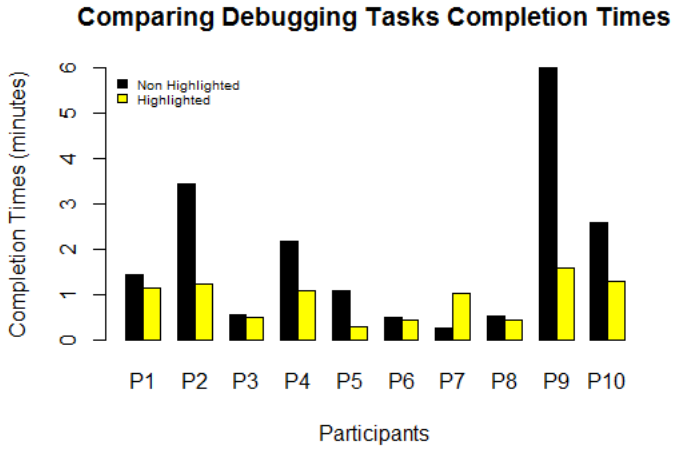

Fig. 5: Bar Graph comparing task completion times for a highlighted writing task with its non-highlighted counterpart

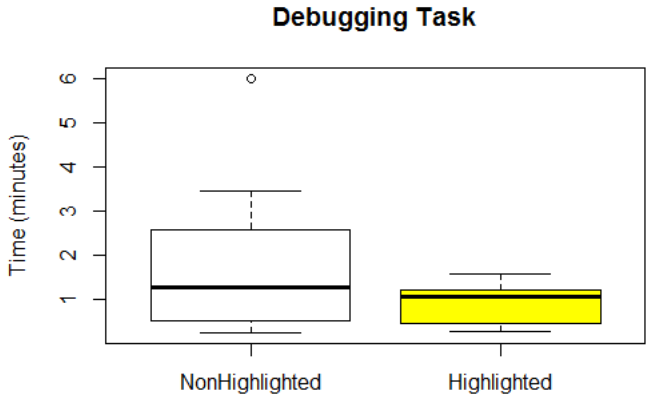

Fig. 6: Boxplots comparing debugging task highlighted and non-highlighted

3.3 Effect of Programming and Musical Experience

To evaluate the effect of programming and musical experience in task completion times performances we introduced a new variable: time advantage (Sarkar, 2015). This variable is the ratio of the task completion time in the non-highlighted and in the highlighted task. For example if a participant completed the non-highlighted version of the task in 60 sec and the highlighted in 30 sec then the time advantage variable is 60/30=2. We investigated the correlation of both programming and musical experience with this variable. For the programming experience the time advantage variable for the writing task is distributed normally. Therefore we performed a Pearson correlation test, obtaining a strong correlation value of -0.67 and a p value of 0.0308. Therefore we reject $H3_0$.

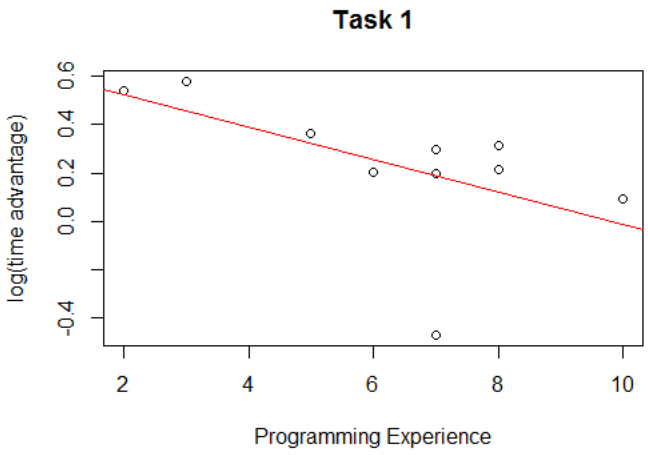

Fig. 7: Each dot is a task instance completed by a participant. The regression line has slope -0.06716

Regarding the debugging task, the Pearson correlation test give a weak negative correlation of -0.2 and the p value shows no significance difference, therefore we cannot reject $H5_0$.
Regarding the musical experience the Pearson test with the writing task shows a positive correlation, but the p value shows no significant difference, therefore we cannot reject $H4_0$. The same happens for the debugging task therefore we cannot reject $H6_0$. It is worth noticing the following. The programming experience plays a fundamental role in decreasing the completion time of the non-highlighted writing task, while the music experience has no effect in the completion time of the non-highlighted task. Qualitatively we could say that musical experience does not affect the speed up performances when the task is non-highlighted, but the programming experience is important to speed up performances when the task is non-highlighted.

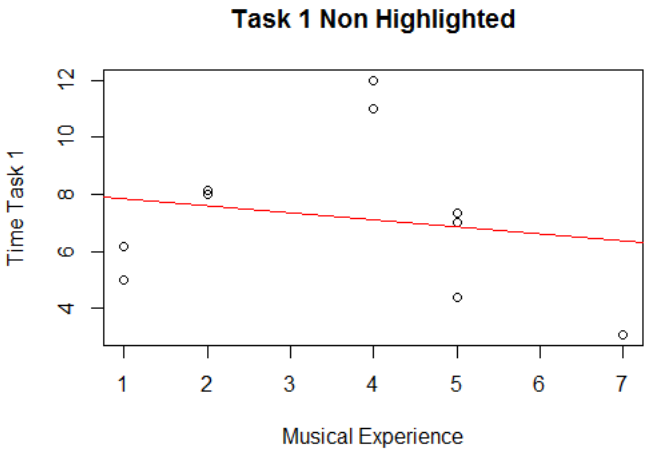

Fig. 8: Task 1 non-highlighted and Musical Experience

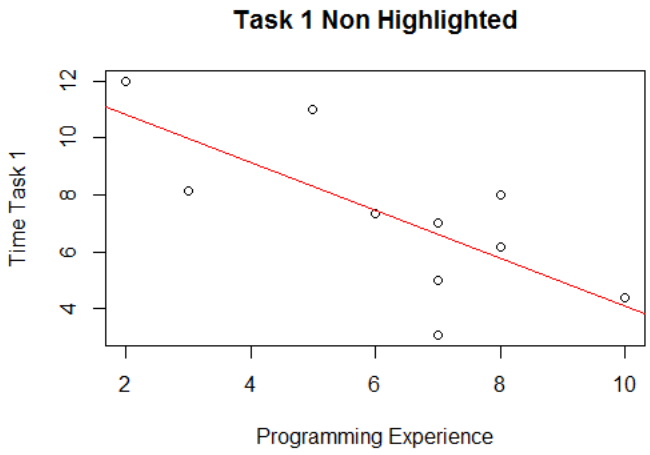

Fig. 9: Task 1 non-highlighted and Programming Experience

4 Discussion

Our results confirm the fact that syntax highlighting improves writing speed and debugging speed of Sonic Pi code. Possibly these effects might increase with the increasing length of the code, and this could be investigated in further research studies. Moreover it is interesting to notice the strong correlation between the programming experience and the impact on the highlighting. A further consideration might be interesting. All the participants were able to complete the task and to let the code run. However some mistakes were still present in the code, and this is due to the fact that also with some syntactic mistakes the code still works, even if the auto completion of the text editor is not available anymore. We suggest that these little bugs maybe should be fixed in a future release of Sonic Pi, also to improve the usability for children learning coding. Moreover another suggestion would be to introduce in the tutorial the same highlighting format that is in Sonic Pi text editor. Finally, it is interesting to add that we executed the experiment also with a 11th participant, a 13 years old child, having interests both in Computer Science and music. The performances reported are on average in line with the ones reported by the 10 University of Cambridge students, and since Sonic Pi is primarily thought for children, this finding seems to confirm the effectiveness of the Sonic Pi approach to coding and music.

5 Conclusion

We investigated the effect, in terms of speed, of syntax highlighting in Sonic Pi, for code writing and debugging. Data were collected from 10 participants. Each participant was requested to write two tasks, in the non-highlighted and in the highlighted version, and to debug a Sonic Pi code in the highlighted and non-highlighted version. The presence of Syntax highlighting significantly reduced the task completion times. Moreover we observed that the task completion times reduces as the experience in terms of programming of the participant increases, possibly showing how the programming experience has a major impact in enhancing the difference between a highlighted and non-highlighted code. Moreover, though much less significant, also music experience has an effect in terms of reduction of completion times in the two tasks of writing and debugging code in Sonic Pi. Future works could investigate these effects in longer programs, or using different types of highlighting with Sonic Pi. Moreover it could be interesting to increase the number of participants to the experiments to investigate further the results obtained. Finally we could focus the data analysis also on other aspects, such as the relationship between musical experience and programming experience.

6 Acknowledgement

Many thanks to Dr. Alan Blackwell, Mariana Mărăşoiu and Hadil Charafeddine for the interesting and important discussions on the topic. Many thanks to all the participants for their valuable time and effort.

References

Aaron, S., & Blackwell, A. (2013). From sonic pi to overtone: Creative musical experiences with domain-specific and functional languages. In *In proceedings of the first acm sigplan workshop on functional art, music, modeling & design.*

Aaron, S., Blackwell, A., & Burnard, P. (in review). The development of sonic pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *The journal of Music Technology and Education,Special Issue Live Coding in Music Education*.

Baecker, R. (1988). Enhancing program readability and comprehensibility with tools for program visualization. In *Software engineering, 1988., proceedings of the 10th international conference on* (pp. 356–366).

Camtasia studio. (n.d.). Retrieved from `http://www.techsmith.com/camtasia.html`

Coding forum 1. (n.d.). Retrieved from `http://tiku.io/questions/3833109/ruby-colon-before-vs-after`

Coding forum 2. (n.d.). Retrieved from `http://www.codecademy.com/forum_questions/51531debf23e2afe38001483`

Coding forum 3. (n.d.). Retrieved from `http://stackoverflow.com/questions/10645668/in-ruby-what-is-the-meaning-of-colon-after-identifier-in-a-hash`

Hakala, T., Nykyri, P., & Sajaniemi, J. (2006). An experiment on the effects of program code highlighting on visual search for local patterns. *Psychology of Programming Interest Group*, 38–52.

Sarkar, A. (2015). The impact of syntax colouring on program comprehension. In *Proceedings of the 26th annual workshop of the psychology of programming interest group (ppig 2015).*

Sonic pi website. (2015). Retrieved from `www.sonicpiliveandcoding.com`

Van Nes, F. (1986). Space, colour and typography on visual display terminals. *Behaviour & Information Technology*, *5*(2), 99–118.

Zmölnig, I., & Eckel, G. (2007). Live coding: An overview. In *Proceedings of the international computer music conference* (Vol. 289).

A Appendix 1-Additional Paper Guideline

WRITING TASK 1

You will have 5 minutes to read the instructions before start writing the task. Feel free to look at the tutorial and the worksheet whenever you want, either while you are reading the instructions and while you are completing the task. Ready for the instructions? Here we go!
You will create the first four bars of a famous children song. In the end you will try to play it and see if you recognize it. If you are able to do it, it means that you followed the instructions in the right way and composed your song with Sonic Pi!
Follow the instructions step by step:
1) Use the synth called fm.
2) Use two blocks of 2.times loop which you have seen during the tutorial phase. Lets call this two blocks A and B
3) In block A you will have to play the notes 60,62,64,60 (in this order)
4) In block B you will have to play the notes 64,65,67 (in this order)
5) Define the parameters for each note:
6) In block A the notes 60,62,64,60 will have a release equal to 1
7) In Block B the notes 64 and 65 will have a release equal to 1 and 67 equal to 2
8) Choose the amplitude for each note in block A
9) Choose the amplitude for each note in block B
10) Insert a pause after each note (also after the last one in each block).
11) All the pauses will have the same length except for the last one in the second block that will have double duration with respect to the others.
You are done!
Now try to Run Sonic Pi.
Can you recognize to which famous children song this first four bars belong to?
Let me know when you have finished the task

SOLUTION TO THE TASK:
Brother John
Welcome to Sonic Pi v
use_synth :fm
2.times do
play 60, amp: 0.6, release: 1
sleep 0.5
play 62, amp: 0.8, release: 1
sleep 0.5
play 64, amp: 0.3, release: 1
sleep 0.5
play 60, amp: 0.9, release: 1
sleep 0.5
end

2.times do
play 64, amp: 0.2, release: 1
sleep 0.5
play 65, amp: 0.4, release: 1
sleep 0.5
play 67, amp: 0.3, release: 2
sleep 1
end

DEBUGGING TASK 2

Debug this piece of code. There are some syntactic mistakes. Find them and correct them. You have a maximum of 10 minutes to complete the task.
If you finish before the time given, let me know!

CODE WITH MISTAKES(the mistakes in the code here are highlighted)

```
use_synth : fm
2.times do
play 60, release:0.5 , amp: 4, pan: -1
sleep 0.5
play 67, release: 0.3, amp: 2, pan :1
sleep 0.5
end
use_synth : saw
play 38, release: 0.1, amp: 3
sleep 0.25
play 50, pan : -1
sleep 0.25
use_synth :prophet
play 57, amp: 4, release:0.3
sleep 0.25
```